REMARKS

This is in full and timely response to the above-identified Office Action. Reexamination and reconsideration in light of the proposed amendments and the following remarks are respectfully requested.

Claims Status

In this response:

Claims 2, 19, 20 and 21 are amended;
Claim 22 is newly added; and
Claims 1-22 are pending in this application.

Rejections under 35 USC § 112

Claims 2, 20 and 21 have been amended to obviate the lack of antecedent basis noted in paragraphs #2-5 of this Office Action.

Rejections under 35 USC § 102

The rejection of claims 1-3, 5-8 and 20-21 as being anticipated by Graham is respectfully traversed.

More specifically, the recitation of: "a method for growing a hot trace in a program during the program's execution" is alleged to be anticipated by the disclosure "rearranging the instruction stream of basic blocks in hot traces will improve program execution" which is found at column 5, lines 30-31.

However, this section of Graham discloses:

> Alternatively, the post processor 26 can **retrieve** the profile information of heavily executed traces during execution of the instrumented program. For background, a trace is a sequence of basic blocks that are typically executed together. Typically, heavily executed traces (also known as hot traces) perform a majority of the data structure

> referencing during execution of a program so that reorganizing data structures and rearranging the instruction stream of basic blocks in hot traces will improve program execution efficiency. (Emphasis added)

Thus, as will be noted, this merely discloses what a "trace" and a "hot trace" are, and contains no disclosure of "growing a hot trace." At best all that is disclosed is the possibility that block rearrangement of heavily executed programs or hot traces, will improve program execution efficiency. There is no specific disclosure of rearrangement during program execution. In fact, the disclosure indicates that a record of the heavily executed traces is "retrieved" by the post processor after the fact. The disclosure preceding this section is also such as to disclose the generation of a report. This reinforces the otherwise uncontradicted nuance of Graham that the examination is executed post operation.

Therefore, the requirement that a trace be "grown during the programs execution" is neither disclosed nor suggested.

Note also that the very initial requirement of the claim is neither disclosed nor suggested. All that has been established is that Graham discusses blocks and the "possible" rearrangement of the same. The nebulous statement that the "improvements" can be produced, is basically non-enabling in that there is neither disclosure nor suggestion of just what these "improvements" may be, or how they may be obtained.

The requirement of "in a dynamic translator", which is totally disjointed from the remainder of the sentence from which it is extracted, is alleged to be disclosed at column 3, line 10. However, this section of Graham discloses that;

> As discussed above, translators such as **compilers** are used to translate programs written in a high-level programming language into a form suitable for execution by, for example, a computer. Typically, when **compiling** a program the compiler initially partitions the program into basic blocks in response to command line options from a driver program associated with the compiler. A front-end

> module of the **compiler** in response to a command line option from the driver, transforms source code (i.e., the high level program) in each basic block into internal code which is an intermediate form of object code. Optimizations are typically performed on the internal code in response to a command line option, and object code is then generated. (Emphasis added)

If *(arguedo),* what was claimed was "a compiler which translates . . . " then this may be relevant. However, since the disclosure is directed to compilers which are, as stated in this section of Graham, merely one form of a translator it is incumbent on the PTO to show that the compiler in this instance is in fact a "dynamic translator" as read in light of the specification, and that this compiler could implement the remainder of the steps set forth in the independent claims (noting that the claim must be read as a whole) before any semblance of *prima facie* "anticipation" can be established.

The step of identifying an initial block is alleged to be anticipated by the disclosure at column 3, 25-26. More specifically, the statement that "a basic block begins with a label . . ." is supposed to disclose in some way the act of "identifying an initial block." More specifically, this section of Graham discloses:

> A basic block of a program is a sequence of instructions that are executed together. That is, if the first instruction of a block is executed, then the other instructions in the block should also be executed. Typically, **a basic block begins with** a label **and ends with** a conditional or unconditional branch instruction or a return instruction. (Emphasis added)

This discusses the structure of a basic block (singular). It is therefore submitted that the step of determining which of a number of blocks is an "initial" block is not disclosed by this section of Graham. That is to say, as noted above, this disclosure is such as to disclose the structure of each block as different from a series of the same. Further, while it is noted that Graham does make reference to a "source basic block"

(note the column 6, lines 41-44 section of Graham quoted below), there is no disclosure of how this "source basic block" is identified per se, i.e., an identification act.

The rejection further states that the claimed branch prediction feature is disclosed at column 6, lines 41-44 and quotes: "the basic blocks of the program can be arranged so that the most frequent path taken at a conditional branch falls through to the next sequential basic block . . ."

However, this section of the Graham reference, discloses:

> A restructuring module 36 receives profile information from, for example, the block execution table and the data structure **record after termination of the instrumented program**. The restructuring module then determines if the data structures can be reorganized so that fewer cache lines are used to improve the execution time of the program. For example, the cache resources may be reallocated so that data structure elements of most frequently used data structure patterns use the same cache line. As noted, **the compiler also restructures the instruction stream of the program**. Preferably, the instructions stream of the program may be restructured so that the most frequently executed paths consists of contiguous basic blocks. That is, the basic blocks of the program can be arranged so that the most frequent path taken at a conditional branch **falls through to the next sequential basic block** as opposed to jumping to a non-sequential successor block. In addition, **unconditional branches** from a **source basic block** to **a destination basic block** that occur in frequently executed paths **can be removed** by, for example, **placing the source and destination basic blocks involved together**. (Emphasis added)

The anticipation of claim 2 is traversed for at least the same reasons advanced above. For example, column 5, lines 30-31, of Graham does not disclose "a method for growing a hot trace in a program during the program's execution" for the same reasons set forth *supra.*

Further, claim 2 does not just call for "branch prediction" it calls for:

> "until an end-of-trace condition is reached, applying static branch prediction rules to the terminating branch of a last block in the trace to identify a next block to be added to the selected trace."

This is different from the limitation that the rejection seeks to assert is anticipated. It simply is not permissible to arbitrarily précis the claim and assert that some vaguely similar disclosure is such as to anticipate the claimed step. Indeed, merely by way of example, there is no disclosure of "static branch prediction rules" in Graham. In fact there is no disclosure of how the rearrangement suggested in Graham is actually achieved.

It is therefore submitted that the mention of branch prediction does not anticipate the limitations contained in the claimed step in question.

Additionally, column 1, lines 6-13 of Graham sets forth that:

> The present application relates to a program analysis method for monitoring and reporting data structure activity of a program. More particularly, the present application profiles basic blocks of a program by monitoring block arcs and **provides reports** on block and data structure activity of the program to tune the performance of the program when translated to executable program code. (Emphasis added)

The "in a dynamic translator" language, is not anticipated by the disclosure at column 3, line 10, for the same reasons advanced *supra.* In fact, it is seen as

untenable to select portions of the claims and reject the same out of context using passages which bear but a semblance of the same disclosure. That is to say, the passage relied upon for rejection basically suggests that the translator can be a compiler and then goes on to discuss the functions of a compiler, rather than dynamic translation.

In addition, in connection with disclosure of Graham at column 6, lines 48-49, which is alleged to anticipate the claimed step of "adding the identified next block to the selected trace", it is noted that placing source and destination blocks together is fundamentally different from adding another block to a sequence of selected blocks. That is to say, placing two particular blocks together is different from what is claimed. More specifically, what is claimed calls for "identifying an initial block" then adding blocks to the trace by "applying static branch prediction rules to a terminating branch of a last block in the trace to identify a next block to be added to the selected trace." This is continued "until an end-of trace condition is reached."

Clearly, with the claimed subject matter there is at the very least, one block between the source and destination blocks and neither a *prima facie* case of anticipation nor obviousness is established by the disclosure of Graham.

In connection with claim 3 it is asserted that "reorganizing data structures of the program to reduce the data cache usage of the program in response to said program activity" anticipates the step of "storing selected traces in a code cache." This is traversed. The Graham disclosure indicates activity to reduce cache usage while the claim specifically requires cache usage to store selected traces. There is essentially no nexus between the two. The claims call for a code cache while the disclosure is silent as to this type of cache.

Claim 5 calls for the prediction rules to include both rules for predicting the outcomes of branch conditions and rules for predicting the targets of branches. The nebulous statement that Graham teaches predicting the outcomes and target of branches does not disclose the use of different rules in the manner specified in claim 5.

It is alleged that the disclosure at column 6, lines 42-44 anticipates the following claim 6 recitation:

> an initial block is identified by maintaining execution counts for targets of branches and when an execution count exceeds a threshold, identifying as an initial block, the block that begins at the target of that branch and extends to the next branch.

However, this section of Graham discloses that:

> the basic blocks of the program can be arranged so that the most frequent path taken at a conditional branch falls through to the next sequential basic block as opposed to jumping to a non-sequential successor block. In addition, unconditional branches from a source basic block to a destination basic block that occur in frequently executed paths can be removed by, for example, placing the source and destination basic blocks involved together.

It is not seen that this section of Graham discloses, for example, "maintaining execution counts" or a threshold against which the count is compared. This section of Graham further fails to disclose any action which is triggered the threshold being exceeded.

Claim 7 calls for the set of prediction rules to comprises: for the branch instruction, determining whether to add a target instruction of the branch instruction to the hot trace based on said set of static branch prediction rules. Column 6, lines 41-45 of Graham cited by the examiner discloses:

> . . . the basic blocks of the program can be arranged so that the most frequent path taken at a conditional branch falls through to the next sequential basic block as opposed to jumping to a non-sequential successor block.

How this is meant to anticipate the subject matter of claim 7 is simply not understood. There is neither disclosure of adding a target instruction nor any decision to consider the same.

The rejections of claims 20 and 21 are traversed on the same basis as above.

Rejections Under 35 USC § 103

The rejections of claims 4 and 9-19 under 35 USC § 103(a) as being unpatentable over Graham taken in view of one or more of Lethin et al., Chang and Sager, or combinations thereof, are traversed. Claim 4 and claims 9-19 depend directly or indirectly from claim 2. However, as noted above, the steps which are recited in claim 2 are not disclosed in Graham and the anticipation rejection of at least claims 2, 3 and 5-8 is untenable. The obviousness rejections therefore fall with the anticipation rejections.

Newly added claim

Claim 22 is added in this response. This claim finds full support in at least claim 1 on which is based. Claim 22 is patentable in that it recites subject matter neither disclosed nor suggested in the art of record.

Conclusion

It is submitted that the claims are now clear and distinct and that this application stands in condition for allowance for at least the reasons advanced above.

Respectfully submitted,

Date March 17, 2004

By _____

William T. Ellis
Registration No. 26,874

Keith J. Townsend
Registration No. 40,358